Robot Code Review

Welcome!

Basic robot code knowledge review

Two phases in each game:

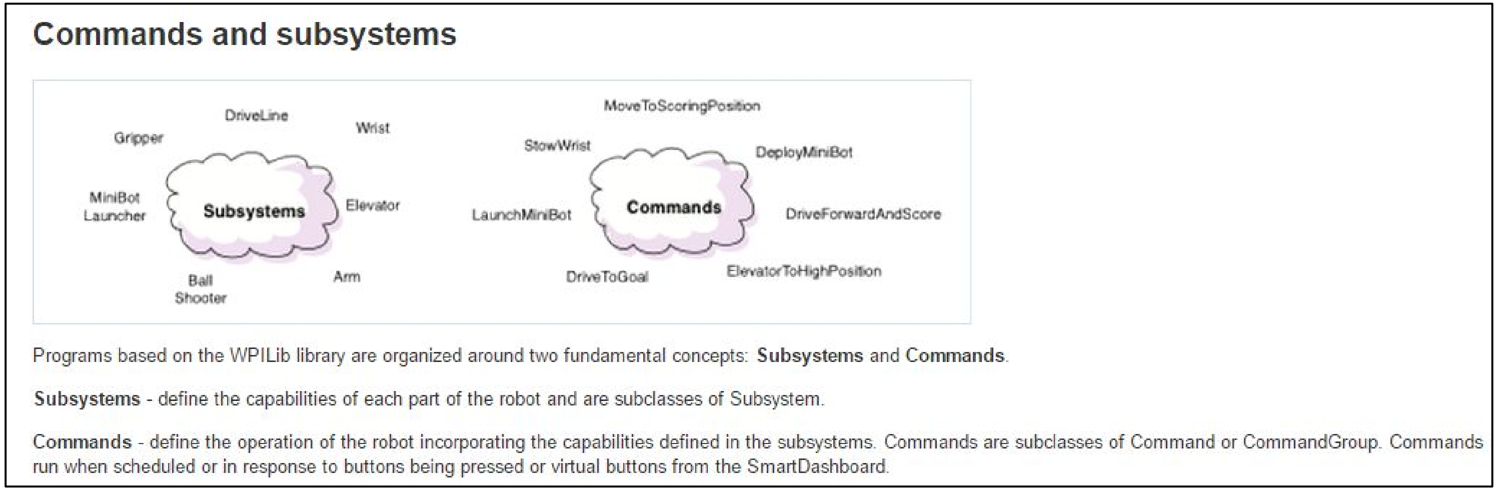
- Autonomous
  - The robot runs on preprogrammed instructions
- Teleoperated
  - The robot runs on commands as given by a driver

Two things we need to account for each year:

- I/O (Input/Output).
  - This is the robot's way of sending and receiving data between the drivers and itself. I/O includes the joysticks, Xbox controller, driver's station, and SmartDashboard.
- Drivetrain: allows the robot to drive. Drivetrains can be:
  - Tank (driven with two joysticks)
  - Mecanum/Omnidirectional (special wheels that can move the robot horizontally and pivot in place, can be driven using tank methods)

What is command based programming?

From the FRC resources:



Commands and subsystems

Programs based on the WPILib library are organized around two fundamental concepts: **Subsystems** and **Commands**.

Subsystems - define the capabilities of each part of the robot and are subclasses of Subsystem.

Commands - define the operation of the robot incorporating the capabilities defined in the subsystems. Commands are subclasses of Command or CommandGroup. Commands run when scheduled or in response to buttons being pressed or virtual buttons from the SmartDashboard.

Command based programming structures code by splitting it into what the robot can do and what it will do. Allows the same subsystem to be used in multiple ways- for example, ShootHighGoal and ShootLowGoal.

How do we write command based code?

- There are three classes automatically made for your: OI, Robot, and RobotMap.
    - OI is where you bind the buttons and joysticks to your commands - the OI class outlines how a robot behaves when a button is pressed.
    - Robot is where you define what the robot does in test, autonomous, and teleop, as well as what it does when it starts up. This includes anything that has to happen initially for the robot to work.
    - RobotMap is where you declare your Device IDs and port numbers. RobotMap connects the physical robot connections and code.

Subsystem

- Can you guess what this subsystem is?
  - package org.usfirst.frc.team2265.robot.subsystems;
  - import org.usfirst.frc.team2265.robot.RobotMap;
  - import edu.wpi.first.wpilibj.CANTalon;
  - import edu.wpi.first.wpilibj.Gyro;
  - import edu.wpi.first.wpilibj.Joystick;
  - import edu.wpi.first.wpilibj.RobotDrive;
  - import edu.wpi.first.wpilibj.command.Subsystem;

Subsystem

- These are some instantiations of the objects used. What do they correspond to?
  - public static CANTalon frontLeft = new CANTalon(RobotMap.frontLeftPort);
  - public static CANTalon rearLeft = new CANTalon(RobotMap.rearLeftPort);
  - public static CANTalon frontRight = new CANTalon(RobotMap.frontRightPort);
  - public static CANTalon rearRight = new CANTalon(RobotMap.rearRightPort);
  - public static RobotDrive mecanumDrive = new RobotDrive(frontLeft, rearLeft, frontRight, rearRight);

  - public static Joystick leftJoy = new Joystick(RobotMap.leftJoyPort);
  - public static Joystick rightJoy = new Joystick(RobotMap.rightJoyPort);

Subsystem

- This gets the values from the joysticks and applies them to the mecanumDrive method.
- Notice how the y, x, and rotation are all controlled separately.
  - public void drive () {
  - driveX = leftJoy.getX();
  - driveY = leftJoy.getY();
  - driveZ = rightJoy.getX();
  - mecanumDrive.mecanumDrive_Cartesian(driveX, driveY, driveZ, gyroscope.getAngle());
  - }

Subsystem

- Accessor methods get and return a number. How would you write an accessor method to return a gyroscope angle? You can use all available resources.

Subsystem

- Write an accessor method that returns the gyroscope angle.
  - public double getDegrees () {
  - return gyroscope.getAngle();
  - }

Subsystem

- Closing the class.

```
public double[] getEncoderValues() {
 double[] values = new double[4];
 values[0] = frontLeft.getEncPosition();
 values[1] = rearLeft.getEncPosition();
 values[2] = frontRight.getEncPosition();
 values[3] = rearRight.getEncPosition();
 return values;
}

  public void initDefaultCommand() {
      // Set the default command for a subsystem here.
      //setDefaultCommand(new MySpecialCommand());
  }
```

Subsystem

- SUMMARY:
  - Subsystems are similar to traditional classes.
  - They outline the functions of a system using methods, but don’t use the methods they create.
  - Subsystems correspond to physical objects (in this case, the chassis.)

Practice

Using all available resources can you create a tank drive?
Tank drive is a drive system that uses two joysticks.

- Create 2 joysticks,
- Create 4 motors,
- Create drivetrain,
- How does one use the tankDrive() method correctly?

Commands

- What does this command do?
  - package org.usfirst.frc.team2265.robot.commands;
  - import org.usfirst.frc.team2265.robot.Robot;
  - import edu.wpi.first.wpilibj.DoubleSolenoid.Value;
  - import edu.wpi.first.wpilibj.command.Command;

Commands

- Notice the “requires”. Robot.piston is an object; what do you think requires does?
  - public class Extend extends Command {
  -     public Extend() {
  -    // Use requires() here to declare subsystem dependencies
  -      //requires the instance from the main robot class
  -      requires(Robot.piston);
  -     }

Commands

- This contains the “meat” of the command – what happens when it executes.
- protected void initialize() {
  - }
  - protected void execute() {
  - //extends the piston if it is retracted
  - if (Robot.piston.get().equals(Value.kReverse_val)) {
  - Robot.piston.extend();
  - //extend is a method in the Piston subsystem.
  - } }

Commands

- To determine if something isFinished or not, we use a conditional. Here, an if/else is used. What else could be used, and what does the if/else do?
  - // Make this return true when this Command no longer needs to run execute()
  - protected boolean isFinished() {
  - //if the piston is extended, return true, else return false until the action has finished
  - if (Robot.piston.get().equals(Value.kForward_val)) {
  - return true;
  - }
  - else {
  - return false;
  - }
  - }

Commands

- Closing the class.
  - // Called once after isFinished returns true
  -     protected void end() {
  -     }
  -     // Called when another command which requires one of the same subsystems is scheduled to run
  -     protected void interrupted() {
  -     }
  - }

Commands

- SUMMARY:
  - Commands use the methods in Subsystems to do actions.
  - Commands are named after verbs, not nouns, and correspond to what the robot does, not what it can do.
  - Commands are the only things directly run in the Robot class.

Robot

- In Robot, first you need to import everything you are going to use. This includes library classes and all your subsystems.
  - package org.usfirst.frc.team2265.robot;

  - import edu.wpi.first.wpilibj.IterativeRobot;
  - import edu.wpi.first.wpilibj.command.Command;
  - import edu.wpi.first.wpilibj.command.Scheduler;
  - import edu.wpi.first.wpilibj.livewindow.LiveWindow;
  - import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
  - import org.usfirst.frc.team2265.robot.commands.ExampleCommand;
  - import org.usfirst.frc.team2265.robot.subsystems.ExampleSubsystem;
  - import org.usfirst.frc.team2265.robot.subsystems.MecanumDrive;
  - import org.usfirst.frc.team2265.robot.subsystems.Piston;

Robot

- Next, declare the subsystems you will use. This includes OI.
  - public static OI oi;
  - public static Piston piston;
  - public static MecanumDrive drive;

Robot

- The first method is robotInit. This is what happens when the robot is turned on.
  - public void robotInit() {
  - oi = new OI(); //instantiates the OI declared above.
  - // instantiate the command used for the autonomous period
  - autonomousCommand = new ExampleCommand();

Robot

- disabledPeriodic should have little in it; there are no controls on what it does.
  - public void disabledPeriodic() {
  - Scheduler.getInstance().run();

Robot

- autonomousInit is what happens after the robot is turned on in autonomous. This could include any calculations that runs before it moves.
  - public void autonomousInit() {
  -         // schedule the autonomous command (example)
  -         if (autonomousCommand != null) autonomousCommand.start();
  -     }
- autonomousPeriodic is what happens during autonomous. Usually this means running a CommandGroup. This robot has no autonomous.
  - public void autonomousPeriodic() {
  -         Scheduler.getInstance().run();

Robot

- teleopInit does anything that needs to be done at the start of teleoperated mode. Usually this is sparse.
  - public void teleopInit() {
  - // This makes sure that the autonomous stops running when
  - // teleop starts running. If you want the autonomous to
  - // continue until interrupted by another command, remove
  - // this line or comment it out.
  - if (autonomousCommand != null) autonomousCommand.cancel();

Robot

- teleopPeriodic runs every 20ms and schedules what happens in teleop. This includes all of the commands.
  - public void teleopPeriodic() {
  - Scheduler.getInstance().run(); /*THIS IS WHAT MAKES COMMAND BASED CODE POWERFUL. The Scheduler sets the robot in motion based on what the code or driver says.*/
  - SmartDashboard.putNumber("Gyro value = ", drive.getGyroAngle());
  - double[] values = drive.getEncoderValues();
  - SmartDashboard.putNumber("Front Left Encoder Values", values[0]);
  - SmartDashboard.putNumber("Rear Left Encoder Values", values[1]);
  - SmartDashboard.putNumber("Front Right Encoder Values", values[2]);
  - SmartDashboard.putNumber("Rear Right Encoder Values", values[4]);
  - }
  -

Robot

- testPeriodic is used when testing functions. This is a quick way of making sure everything works as intended.

```
public void testPeriodic() {
–           LiveWindow.run();
–     }
– }
```

RobotMap

- Instead of writing a DeviceID number, it is assigned to a constant variable in RobotMap. This makes it much easier to change these IDs, as it is changed only in one place instead of multiple.

RobotMap

- package org.usfirst.frc.team2265.robot;

- public class RobotMap {
- //Joystick Ports
- public static int leftJoyPort = 0;
- public static int rightJoyPort = 1;
- public static int atkJoyPort = 2;
-
- //CAN Talon IDs
- public static int rearLeftPort = 0;
- public static int rearRightPort = 1;
- public static int frontLeftPort = 2;
- public static int frontRightPort = 3;
-
- //Sensors
- public static int gyroPort = 0; //analog
- public static int sol1Port = 0;

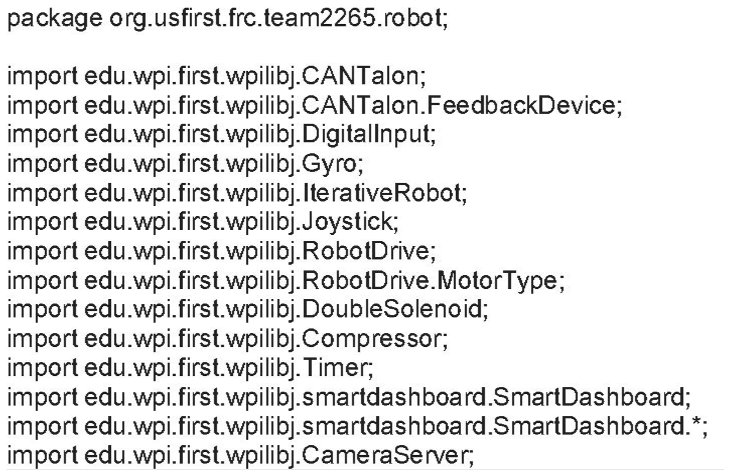
Last year’s code

Features

- Attempted command based code, used iterative.
- One class

Code Review: 2015 Robot Code

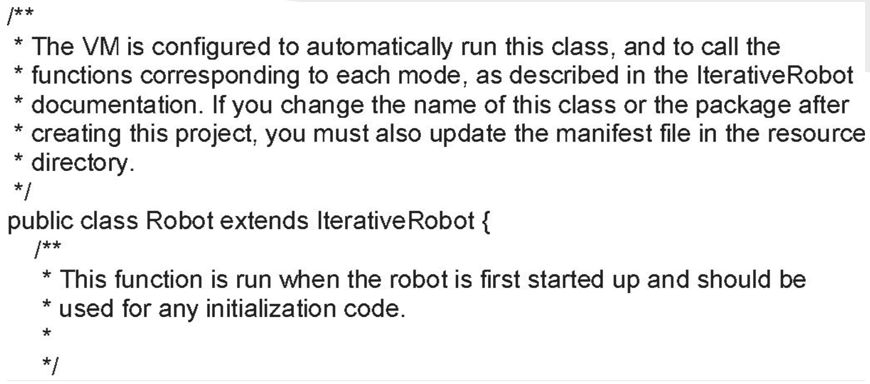
- What does the first line do? How is it different from the rest? What do they do?
- The first line tells the computer to use the FRC package, letting us use the "premade" commands.
- The remaining lines import the classes from which we'll take the commands we use.



```
package org.usfirst.frc.team2265.robot;

import edu.wpi.first.wpilibj.CANTalon;
import edu.wpi.first.wpilibj.CANTalon.FeedbackDevice;
import edu.wpi.first.wpilibj.DigitalInput;
import edu.wpi.first.wpilibj.Gyro;
import edu.wpi.first.wpilibj.IterativeRobot;
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.RobotDrive.MotorType;
import edu.wpi.first.wpilibj.DoubleSolenoid;
import edu.wpi.first.wpilibj.Compressor;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard.*;
import edu.wpi.first.wpilibj.CameraServer;
```

Code Review: 2015 Robot Code

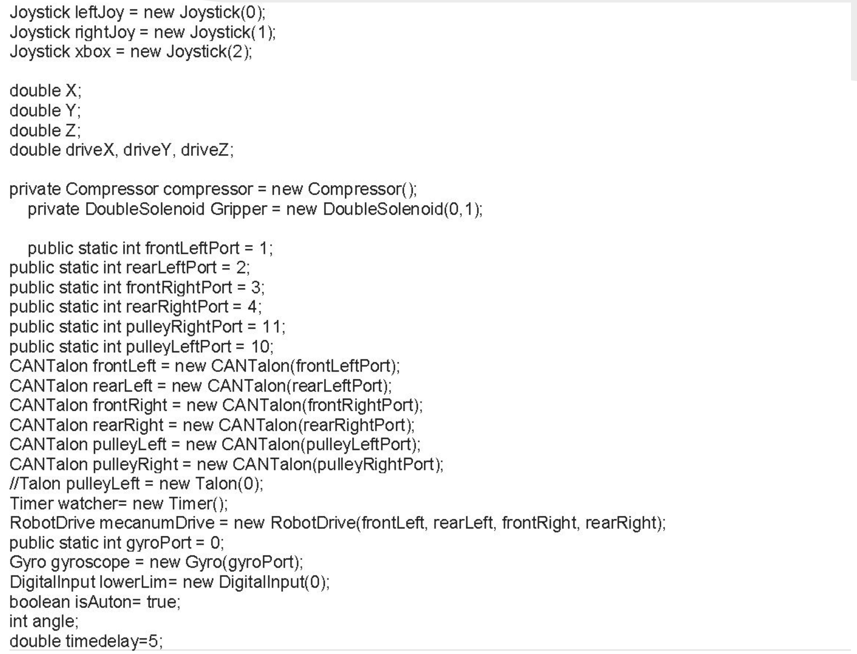
- The only line that matters here is “public class Robot extends IterativeRobot {“, because it lets the computer know that we’re writing in the Iterative format. Everything else is for the benefit of the reader, so it’s commented out.



```
/**
 * The VM is configured to automatically run this class, and to call the
 * functions corresponding to each mode, as described in the IterativeRobot
 * documentation. If you change the name of this class or the package after
 * creating this project, you must also update the manifest file in the resource
 * directory.
 */
public class Robot extends IterativeRobot {
    /**
     * This function is run when the robot is first started up and should be
     * used for any initialization code.
     *
     */
```

Code Review: 2015 Robot Code

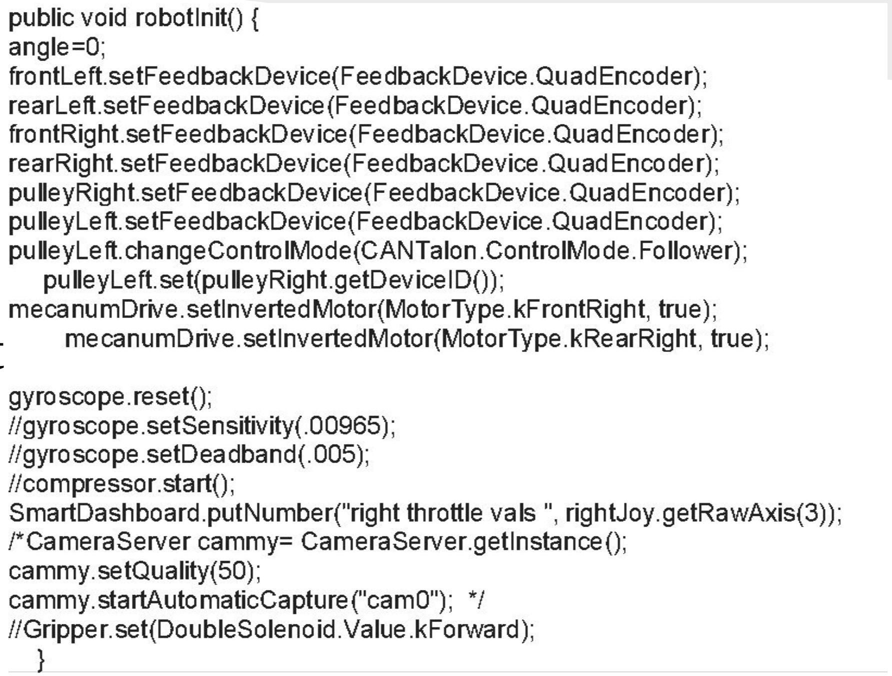
- This portion of the code declares variables that will be used. Declarations do not have to set the variable's value, but they often do.



```
Joystick leftJoy = new Joystick(0);
Joystick rightJoy = new Joystick(1);
Joystick xbox = new Joystick(2);

double X;
double Y;
double Z;
double driveX, driveY, driveZ;

private Compressor compressor = new Compressor();
    private DoubleSolenoid Gripper = new DoubleSolenoid(0,1);

    public static int frontLeftPort = 1;
public static int rearLeftPort = 2;
public static int frontRightPort = 3;
public static int rearRightPort = 4;
public static int pulleyRightPort = 11;
public static int pulleyLeftPort = 10;
CANTalon frontLeft = new CANTalon(frontLeftPort);
CANTalon rearLeft = new CANTalon(rearLeftPort);
CANTalon frontRight = new CANTalon(frontRightPort);
CANTalon rearRight = new CANTalon(rearRightPort);
CANTalon pulleyLeft = new CANTalon(pulleyLeftPort);
CANTalon pulleyRight = new CANTalon(pulleyRightPort);
//Talon pulleyLeft = new Talon(0);
Timer watcher= new Timer();
RobotDrive mecanumDrive = new RobotDrive(frontLeft, rearLeft, frontRight, rearRight);
public static int gyroPort = 0;
Gyro gyroscope = new Gyro(gyroPort);
DigitalInput lowerLim= new DigitalInput(0);
boolean isAuton= true;
int angle;
double timedelay=5;
```

Code Review: 2015 Robot Code

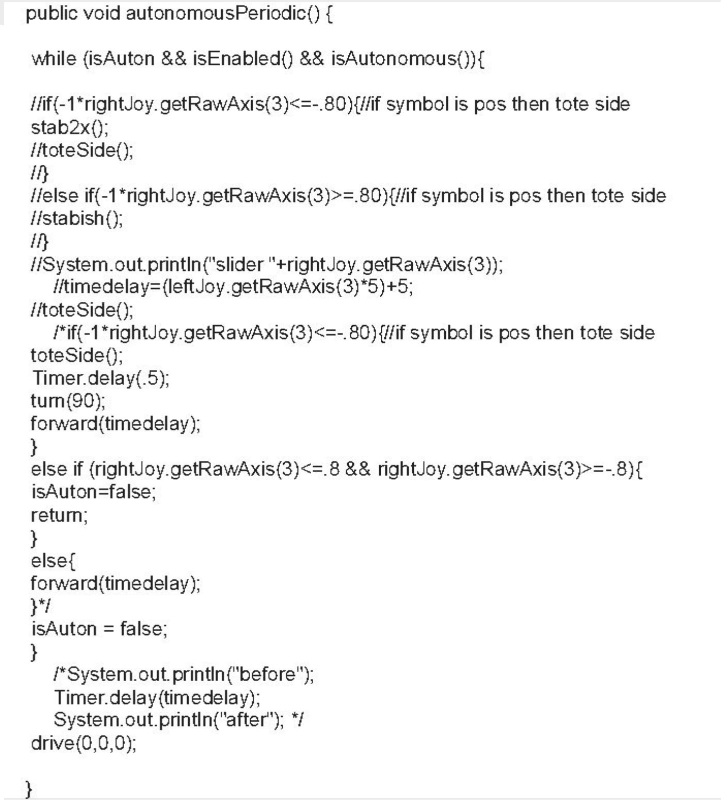
- This is inside the Robot class, so it runs on initialization.
- setFeedbackDevice means that the QuadEncoder will report data from that Talon.
- Gyroscope.reset sets the current angle/heading to 0.



```
public void robotInit() {
angle=0;
frontLeft.setFeedbackDevice(FeedbackDevice.QuadEncoder);
rearLeft.setFeedbackDevice(FeedbackDevice.QuadEncoder);
frontRight.setFeedbackDevice(FeedbackDevice.QuadEncoder);
rearRight.setFeedbackDevice(FeedbackDevice.QuadEncoder);
pulleyRight.setFeedbackDevice(FeedbackDevice.QuadEncoder);
pulleyLeft.setFeedbackDevice(FeedbackDevice.QuadEncoder);
pulleyLeft.changeControlMode(CANTalon.ControlMode.Follower);
    pulleyLeft.set(pulleyRight.getDeviceID());
mecanumDrive.setInvertedMotor(MotorType.kFrontRight, true);
      mecanumDrive.setInvertedMotor(MotorType.kRearRight, true);

gyroscope.reset();
//gyroscope.setSensitivity(.00965);
//gyroscope.setDeadband(.005);
//compressor.start();
SmartDashboard.putNumber("right throttle vals ", rightJoy.getRawAxis(3));
/*CameraServer cammy= CameraServer.getInstance();
cammy.setQuality(50);
cammy.startAutomaticCapture("cam0");  */
//Gripper.set(DoubleSolenoid.Value.kForward);
   }
```

Code Review: 2015 Robot Code

- This is the autonomous method.
- The while loop applies to the whole method and makes sure the robot doesn’t try to do the autonomous method during teleop.
- Notice the multi-line comment starting on “/*if”. Where does it end and what does autonomous do?



```
public void autonomousPeriodic() {

 while (isAuton && isEnabled() && isAutonomous()){

 //if(-1*rightJoy.getRawAxis(3)<=-.80){//if symbol is pos then tote side
 stab2x();
 //toteSide();
 //}
 //else if(-1*rightJoy.getRawAxis(3)>=.80){//if symbol is pos then tote side
 //stabish();
 //}
 //System.out.println("slider "+rightJoy.getRawAxis(3));
    //timedelay=(leftJoy.getRawAxis(3)*5)+5;
 //toteSide();
    /*if(-1*rightJoy.getRawAxis(3)<=-.80){//if symbol is pos then tote side
 toteSide();
 Timer.delay(.5);
 turn(90);
 forward(timedelay);
 }
 else if (rightJoy.getRawAxis(3)<=.8 && rightJoy.getRawAxis(3)>=-.8){
 isAuton=false;
 return;
 }
 else{
 forward(timedelay);
 }*/
 isAuton = false;
 }
    /*System.out.println("before");
    Timer.delay(timedelay);
    System.out.println("after"); */
 drive(0,0,0);

}
```

Code Review: 2015 Robot Code

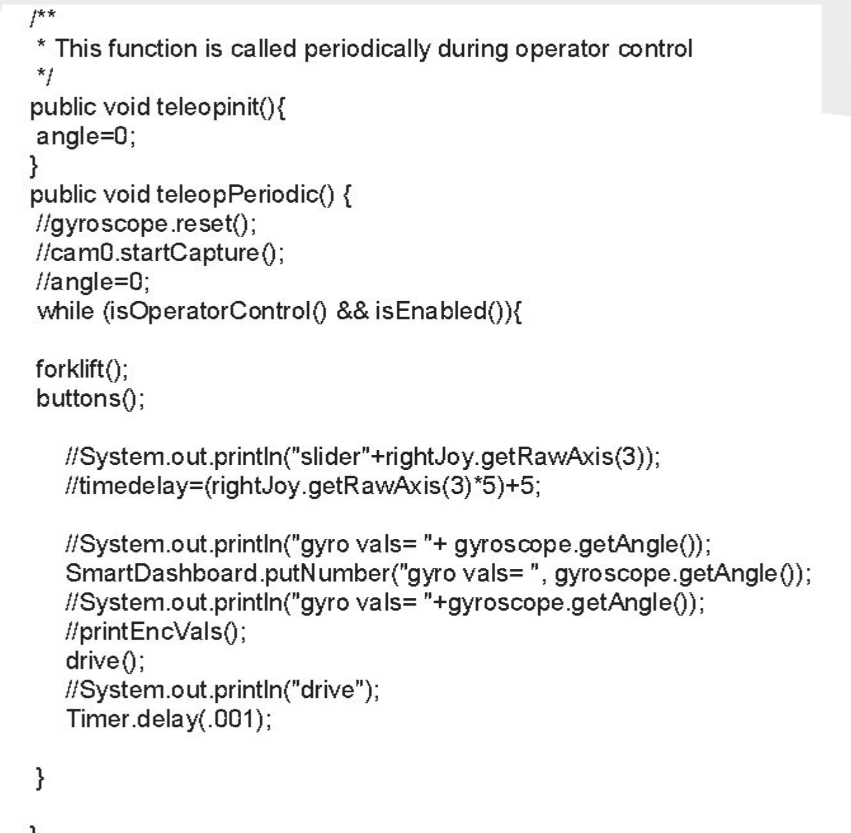
- What do the methods do? When are they called and how can you tell?

  teleopinit() sets the gyroscope to 0, meaning the direction the robot faces is now forward.

  teleopPeriodic is called every .001 seconds + processing time, and excecutes methods forklift() , buttons(), and drive().

- Why do you think Timer.delay is there?

  Timer.delay allows the system to catch up to what we want it do to, meaning it's able to respond to each in the Scheduler and not fall behind.



```
/**
 * This function is called periodically during operator control
 */
public void teleopinit(){
 angle=0;
}
public void teleopPeriodic() {
 //gyroscope.reset();
 //cam0.startCapture();
 //angle=0;
 while (isOperatorControl() && isEnabled()){

 forklift();
 buttons();

    //System.out.println("slider"+rightJoy.getRawAxis(3));
    //timedelay=(rightJoy.getRawAxis(3)*5)+5;

    //System.out.println("gyro vals= "+ gyroscope.getAngle());
    SmartDashboard.putNumber("gyro vals= ", gyroscope.getAngle());
    //System.out.println("gyro vals= "+gyroscope.getAngle());
    //printEncVals();
    drive();
    //System.out.println("drive");
    Timer.delay(.001);

 }
```

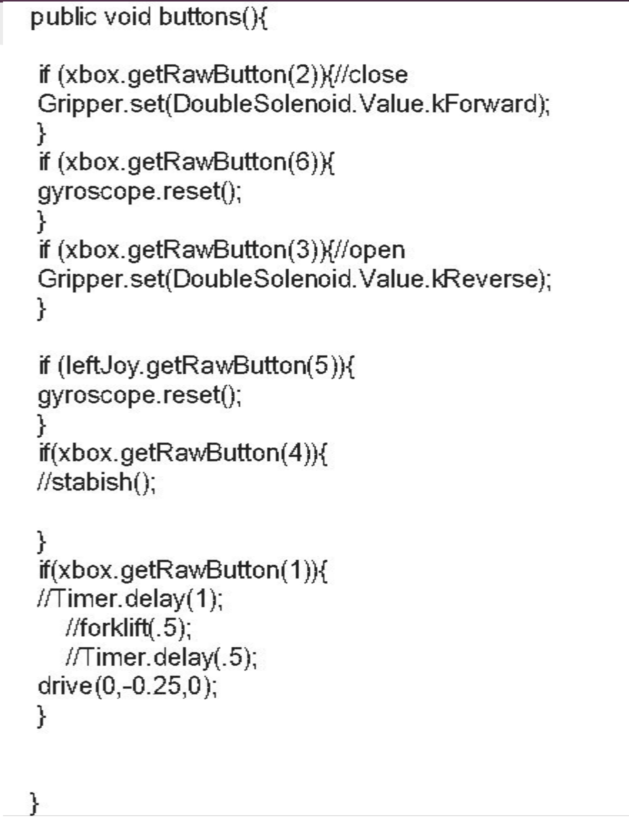
Code Review: 2015 Robot Code

- What do the buttons on the controller do?

  Button 2: Opens and closes the gripper

  Button 6 on the xbox controller and button 5 on the joystick:
    Resets the gyroscope value to zero.

  Button 1: Drives backwards at quarter speed.

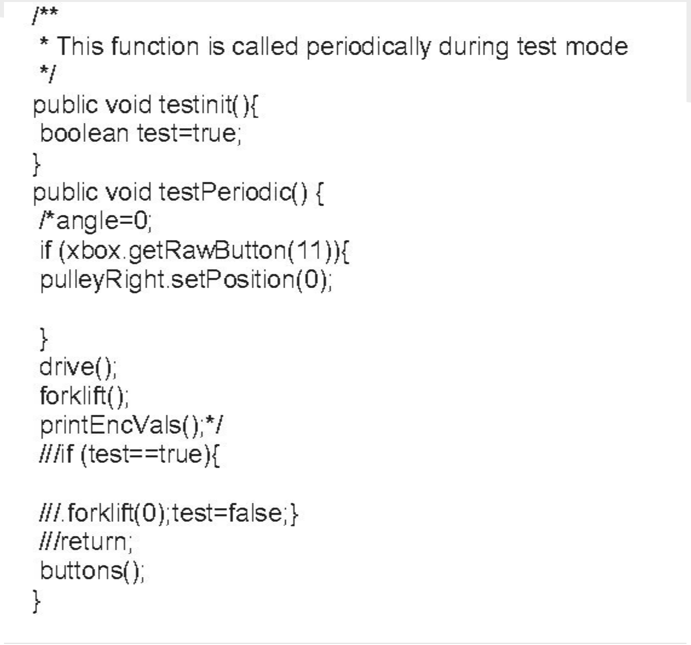


```
public void buttons(){

 if (xbox.getRawButton(2)){//close
 Gripper.set(DoubleSolenoid.Value.kForward);
 }
 if (xbox.getRawButton(6)){
 gyroscope.reset();
 }
 if (xbox.getRawButton(3)){//open
 Gripper.set(DoubleSolenoid.Value.kReverse);
 }

 if (leftJoy.getRawButton(5)){
 gyroscope.reset();
 }
 if(xbox.getRawButton(4)){
 //stabish();

 }
 if(xbox.getRawButton(1)){
 //Timer.delay(1);
    //forklift(.5);
    //Timer.delay(.5);
 drive(0,-0.25,0);
 }


}
```

Code Review: 2015 Robot Code

What does this test?

testinit() sets test to true. testPeriodic() tests to see if the driving, forklift raising and lowering (using pulleys),encoders, and buttons work. testPeriodic() is called continuously.



```
/**
 * This function is called periodically during test mode
 */
public void testinit(){
 boolean test=true;
}
public void testPeriodic() {
 /*angle=0;
 if (xbox.getRawButton(11)){
 pulleyRight.setPosition(0);

 }
 drive();
 forklift();
 printEncVals();*/
 ///if (test==true){

 ///.forklift(0);test=false;}
 ///return;
 buttons();
}
```

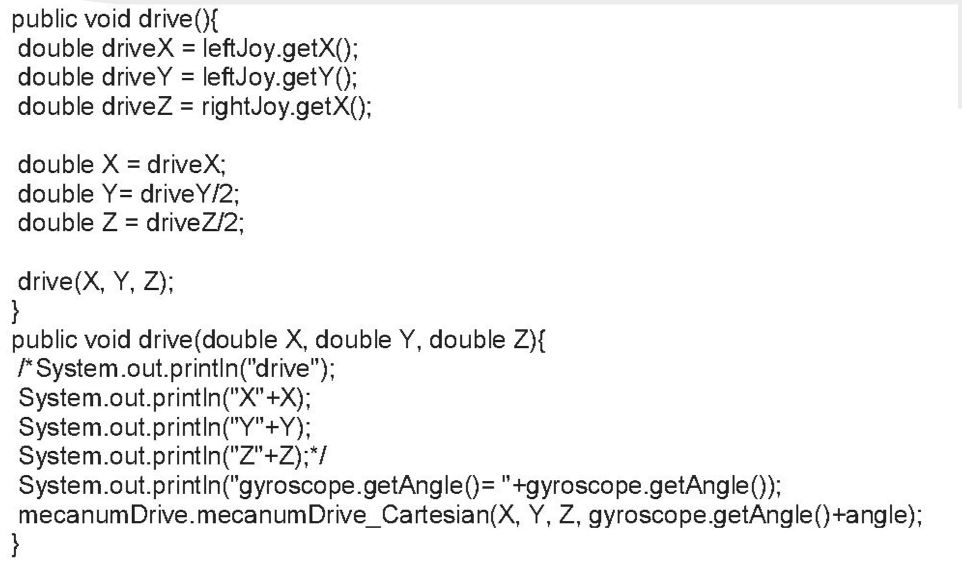
Code Review: 2015 Robot Code

- Where do driveX, driveY, and driveZ come from?

  driveX and driveY are taken from the X and Y axes of the left joystick. driveZ is the value of the x axis of the right joystick.

- Why do you think driveY and driveZ were halved?

  During testing, we found that driveY and driveZ were too sensitive and had to calibrate them.



```
public void drive(){
 double driveX = leftJoy.getX();
 double driveY = leftJoy.getY();
 double driveZ = rightJoy.getX();

 double X = driveX;
 double Y= driveY/2;
 double Z = driveZ/2;

 drive(X, Y, Z);
}
public void drive(double X, double Y, double Z){
 /*System.out.println("drive");
 System.out.println("X"+X);
 System.out.println("Y"+Y);
 System.out.println("Z"+Z);*/
 System.out.println("gyroscope.getAngle()= "+gyroscope.getAngle());
 mecanumDrive.mecanumDrive_Cartesian(X, Y, Z, gyroscope.getAngle()+angle);
}
```

- What does the log show when driving?

  The log shows the X, Y, and Z values of the robot, as well as the angle it is moving in relation to where it was the last time the gyroscope was reset.

- What input does the mecanumDrive method need?

  The X, Y, and Z values the driver desires and current angle on the field.

Code Review: 2015 Robot Code

- What controls the forklift?

  The "RawAxis(1)" on the Xbox controller, or the left joystick on the Xbox controller.

- How does the robot know which way the driver wants the forklift to move?

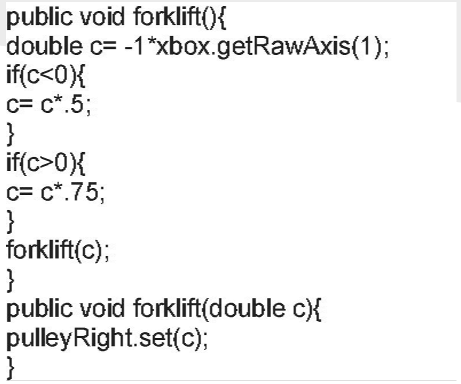
  If the value 'c' of the forklift is positive (pushed forward), the pulleys bring the forklift up. If it is negative (pulled backward), the forklift is sent down.

- Why do you think going up and going down are different speeds?

  Bringing objects up means that dropping them by going too fast would be a problem; if the forklift is going down, dropping an object is not a concern.

- BONUS: Think back to the second slide of code. Why is only pulleyRight set in forklift()?

  pulleyLeft follows whatever speed pulleyRight is on.



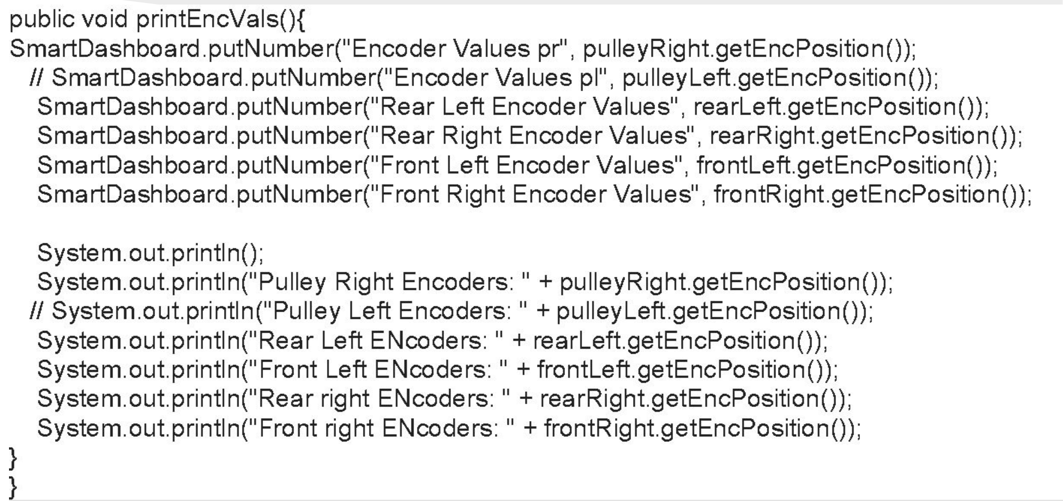
```
public void forklift(){
double c= -1*xbox.getRawAxis(1);
if(c<0){
c= c*.5;
}
if(c>0){
c= c*.75;
}
forklift(c);
}
public void forklift(double c){
pulleyRight.set(c);
}
```

Code Review: 2015 Robot Code

- What is shown on the SmartDashboard and log?

  The encoder values of the wheels and the pulley.

- Why would this be useful to a driver?



```
public void printEncVals(){
SmartDashboard.putNumber("Encoder Values pr", pulleyRight.getEncPosition());
  // SmartDashboard.putNumber("Encoder Values pl", pulleyLeft.getEncPosition());
   SmartDashboard.putNumber("Rear Left Encoder Values", rearLeft.getEncPosition());
   SmartDashboard.putNumber("Rear Right Encoder Values", rearRight.getEncPosition());
   SmartDashboard.putNumber("Front Left Encoder Values", frontLeft.getEncPosition());
   SmartDashboard.putNumber("Front Right Encoder Values", frontRight.getEncPosition());

   System.out.println();
   System.out.println("Pulley Right Encoders: " + pulleyRight.getEncPosition());
  // System.out.println("Pulley Left Encoders: " + pulleyLeft.getEncPosition());
   System.out.println("Rear Left ENcoders: " + rearLeft.getEncPosition());
   System.out.println("Front Left ENcoders: " + frontLeft.getEncPosition());
   System.out.println("Rear right ENcoders: " + rearRight.getEncPosition());
   System.out.println("Front right ENcoders: " + frontRight.getEncPosition());
}
}
```

Your thoughts

What do you think was good about last year's code? What would you change?

GOOD	CHANGE

Homework

Watch the 2015 game animation and write some basic pseudocode of what commands and subsystems you would use if you were writing the code. Email me if you want any help!

How do we write command based code?

- In terms of syntax, command based code looks the same - with loops, methods, classes, variables, parameters, etc.
- Every new command is a new class that extends Command, and every new subsystem extends Subsystem.